Maleku: an evolutionary visual software analytics tool for providing insights into software evolution

Maleku: an evolutionary visual software analytics tool for providing insights into software evolution

Antonio González-Torres, Roberto Therón
and Francisco J. García-Peñalvo
Department of Computer Sciences
University of Salamanca, Spain
Email: {agtorres, theron, fgarcia}@usal.es

Michel Wermelinger and Yijun Yu
Department of Computing
The Open University, UK
Email: {m.a.wermelinger, y.yu}@open.ac.uk

***Abstract*—Software maintenance is a complex process that requires the understanding and comprehension of software project details. It involves the understanding of the evolution of the software project, hundreds of software components and the relationships among software items in the form of inheritance, interface implementation, coupling and cohesion. Consequently, the aim of evolutionary visual software analytics is to support software project managers and developers during software maintenance. It takes into account the mining of evolutionary data, the subsequent analysis of the results produced by the mining process for producing evolution facts, the use of visualizations supported by interaction techniques and the active participation of users. Hence, this paper proposes an evolutionary visual software analytics tool for the exploration and comparison of project structural, interface implementation and class hierarchy data, and the correlation of structural data with metrics, as well as socio-technical relationships. Its main contribution is a tool that automatically retrieves evolutionary software facts and represent them using a scalable visualization design.**

I. Introduction

Software developers and managers often are faced with the maintenance of large legacy applications and software projects to which they usually do not provide support, either within their company or a client company. Consequently, the understanding of the evolution of such legacy applications or software projects is a crucial task for software maintenance. Nevertheless, the maintenance process is usually compromised due to the lack of proper system documentation, which frequently is incomplete, outdated or it is not present [1].

This research focuses on the application of evolutionary visual software analytics to the understanding of software systems. It describes Maleku, a tool that automatically retrieves software evolution details from software repositories and produces software evolution facts that are depicted using software evolution visualization techniques.

Maleku uses two views: a matrix-like representation and a social-network graph. The matrix view is a scalable and interactive visualization for showing the correlation of structural data with metrics, programmer contributions, the creation of software items, socio-technical relationships, and inheritance and interface implementation relationships.

Evolutionary visual software analytics is a special application of visual analytics, and even more specifically of visual software analytics, to software evolution. Accordingly, evolutionary visual software analytics uses as reference the visual analytics mantra [2], [3].

The remainder of the paper is organized as follows: section II describes the tool architecture, while section III explains the designs of the visual representations used by Maleku, section IV briefly describes some scenarios when using the tool, section V briefly discuss some related works and finally section VI presents the conclusions.

II. Architecture

Figure 1 shows the architecture of Maleku. Such architecture is based on the evolutionary visual software analytics process and it consist of five components: the knowledge extraction engine on a evolutionary basis, the software evolution facts, the data transformation engine for visualization models, the software evolution visualizations and the user itself.

The knowledge extraction engine on a evolutionary basis supports the addition of modules for supporting new software configuration management systems (SCM) and allows configuring connections to several different projects and software repositories for extracting evolutionary details and carrying out software evolution analysis. Its components are the monitor of new revisions, the source code extractor, the source code parser and the metadata and software evolution analysis engine. The monitor of new revisions is a process that continuously monitors the addition of new revisions to software projects and notifies about new revisions to the source code extractor. Then, the source code extractor starts extracting source code from the software repository and invokes the source code parser for collecting details about the software project structure, the hierarchy of classes, the coupling relationships and the source code and metrics raw data for calculating metrics for classes and methods. Finally, the metadata and software evolution analysis engine takes the outcomes produced by the source code parser and query additional details from the software repository. Then, it applies an exhaustive software evolution analysis and stores the results in the software evolution facts and structure evolution database.

The software evolution facts that have been taken into account for the visualizations presented later are the software item lifelines, the evolution metrics, the socio-technical relationships and some architectural/structural relationships such

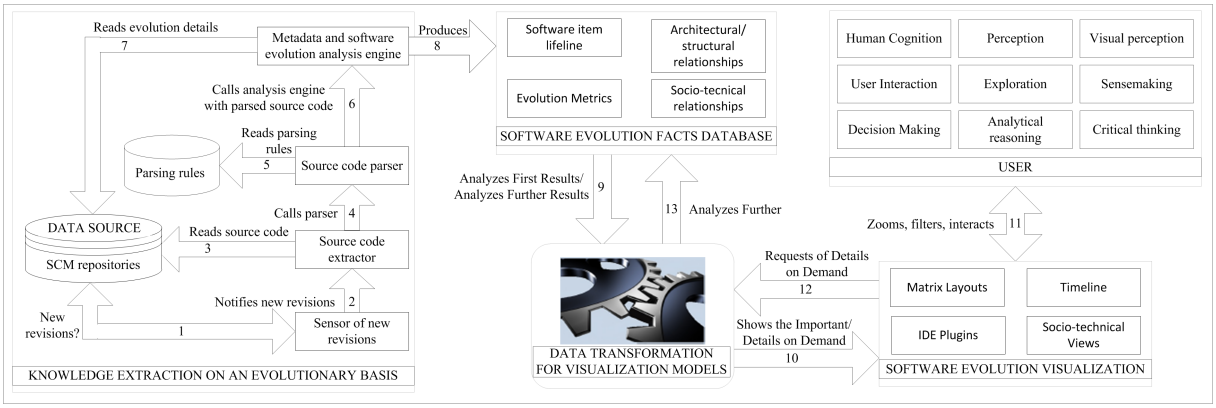

Fig. 1. Evolutionary visual software analytics architecture of Maleku

as inheritance, interface implementation and the correlation of structural data with metrics. Therefore, the data transformation engine for visualization models transforms the software evolution facts data structures into the appropriate data structures that are used by software evolution visualizations. Therefore, the visualizations included in Maleku are embedded into an Eclipse plugin and make use of a matrix-like representation, a timeline and socio-technical views that are supported by a social-network graph and the use of colors for representing programmer's contributions.

The knowledge extraction engine and the software evolution facts database are server side components with a graphical interface. While the data transformation engine is a middleware process that is configured using a graphical interface and is executed automatically when it detects new additions to the software evolution facts database.

III. Visual representations

A matrix-like representation has been chosen as the main view for being a simple structure widely known by programmers and because it allows establishing relationships among multivariate elements easily. The cells of the matrix representation, with the packages and software items in the rows and the time units in the columns, are used for the correlation of details associated to the corresponding package or software item. Basically, the evolution details that are correlated with the project structure are programmer contributions, the creation of software items, the addition or removal of inheritance or interface implementation relationships and software item metrics.

The interactions supported by the matrix view include the possibility of zoom-in and zoom-out, the fisheye distortion, the reorder of project structure elements and the capability of filtering out nodes from the structure. In addition, it supports year selection from the timeline for depicting data according to associated months and in the socio-technical view. Moreover, the user has the possibility to choose how metrics and programmers are represented by selecting between relative or absolute value representations.

The visual representation of the project structure is made up of all the packages and software items that have been added to the project during its evolution. This allows to correlate all software items involved in the evolution process with programmers, metrics and architectural relationships such as inheritance and interface implementation relationships. Figure 3(a) shows on the left side the current project structure of jEdit in the Eclipse workbench, while on the right side is displayed the visual representation depicting the project and its corresponding structure, including packages and software items that are no longer part of the current project version. Additionally, Figure 3(b) illustrates that source code files could be expanded for showing the software items they contain. Such design feature allows to depict the lifeline of software items and packages using an intuitive approach, as shown in Figure 2. The details view on the right shows that the performed activities on the file macos.Delegate.java were carried out between 2003 and 2008. Moreover, those packages currently are not part of the latest version of the project, which is corroborated when reviewing the project structure on the Eclipse workbench, as shown in Figure 3(a). So, the lifeline of a package or software item is determined by the representation of programmer activity in the matrix view.

Another visualization key feature is the representation of inheritance and interface implementation, which is depicted in Figure 3(c). Foldable tree nodes are used for representing software item inheritance and implementation relationships. Figure 4 shows that the establishment of inheritance and interface implementation relationships are depicted by a green oval and its termination is represented by a red oval. In addition, the location of associated software items is explicitly indicated (Java, current project or external library).

Software item metrics are represented using bar charts with the aim of highlighting changes (see Figure 5). Similar to the representation of programmer contributions, metrics are represented using relative and absolute areas. Relative representation takes into consideration the software item with the highest metric value for calculating the chart height, while the absolute representation only takes into consideration the highest metric value associated to the software item.

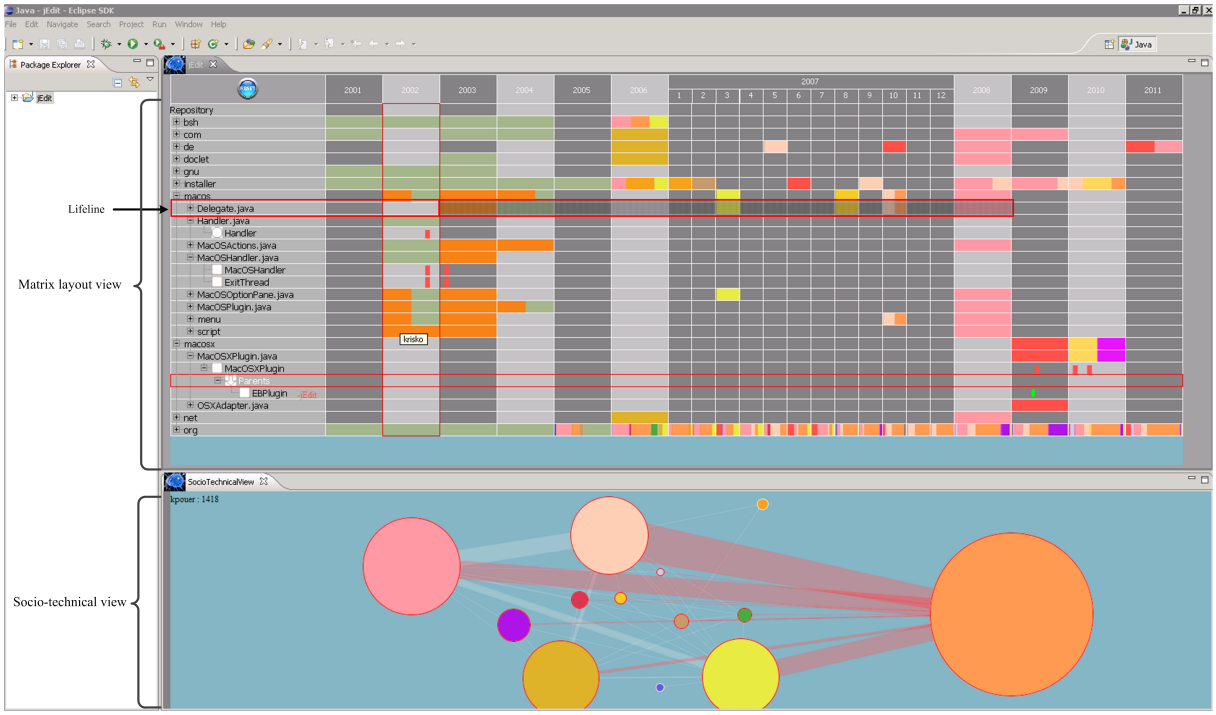

Fig. 2. Main view of the visualizations

The other included view in the visual representation design is the socio-technical view (see Figure 2), which is subordinated to the matrix view. When users select a time unit in the matrix view the associated data are loaded into the socio-technical view. Loaded data represent the socio-technical relationships that are derived from the modification of software items. Such relationships are established from the software items that programmers have modified in common. The nodes of the graph represent to programmers and their size the number of contributions they have been made. In addition, edges represent the relationships among programmers and their thickness represent the number of software items they have modified in common. Node colors represent programmers and are associated to the colors used in the matrix view.

IV. Scenarios

The visual representations included in Maleku have been integrated into Eclipse as a plugin, shown in Figure 2.

The interaction path, followed by users, starts with the selection, from a popup menu, of the view that they want to display in the Eclipse workbench. From this point users interact with the visualization and the interaction path branches. So, several scenarios are possible according to maintenance needs. The following scenarios are some practical examples in which Maleku visualizations could be applied.

Scenario 1: A bug has been reported and the project manager has to assign a programmer to solve it. Once the project manager has analyzed who is the most suitable programmer for solving the reported issue, the project manager realizes that the developer in charge of the components in which the problem is localized is on vacations. Then, the project manager follows the next steps:

1) Open the matrix view for the complete project or a given package and then it opens the socio-technical view.
2) Select a recent time unit, with a considerable number of activities, from the timeline in the matrix view.
3) The socio-technical view is updated with the data associated to the selected time unit.
4) The project manager selects the programmer that is in a sick leave from the socio-technical view and its relationships with programmers that have changed the same highlighted software items.
5) Then, the project manager determines which programmer is the most suitable for solving the opened issue.

Scenario 2: Programmers are in the middle of a software project refactoring and they have to review how changes made by other programmers affect inheritance and interface implementation relationships. Programmers follow the next steps after opening the matrix view for the complete project or a given interesting package:

1) Choose the source code file that contains the software item in which the programmer is interested and expand it to review the relationships of such software item.
2) Expand the software item and examine its inheritance and interface implementation relationships to determine if changes have occurred: which relationships have been established and which have been terminated.

The tool and video demos are downloadable from analiticavisual.com/maleku.

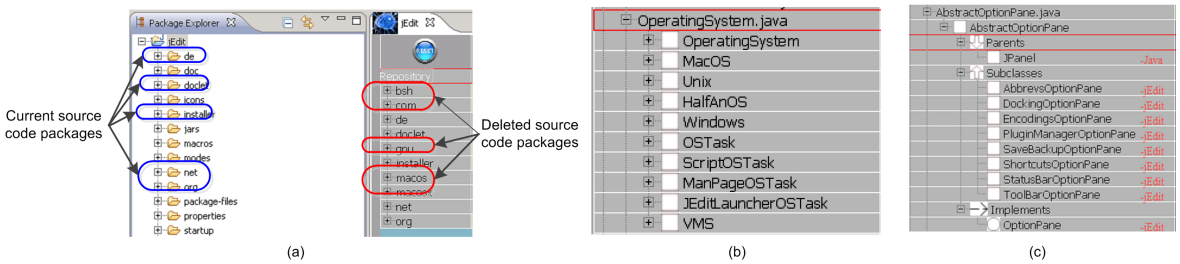

Fig. 3. Project structure and file contents. (a) Project structure representation. (b) Software items contained by a file. (c) Inheritance and interface implementation relationships.

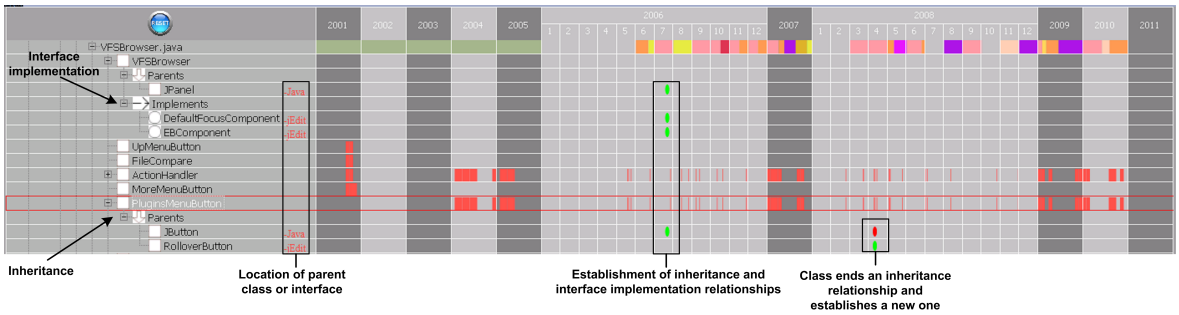

Fig. 4. Inheritance and interface implementation relationships, including expanded years and metric values.

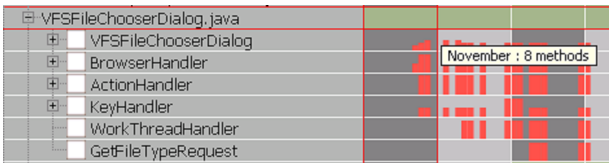

Fig. 5. Metrics representation.

V. Related works

The use of timelines in software evolution visualization is often linked to the representation of software project hierarchies shown with treemaps, graphs, radial layouts and cone trees. Two related approaches to our work that have been applied to visualize software project hierarchies are [4] and [5]. At a higher level of granularity are the evolution visualizations of Pinzger et al. [6] and Voinea et al. [7], which scalably show the evolution of metrics over thousands of software artifacts and hundreds of changes.

VI. Conclusions

The design has taken into account several needs to be informed about the evolution of software projects. Some concern which programmers are participating in the development of specific part of a project, others concern when the project has taken place and when the solution becomes stable. It also can inform the software maintainers about which programmer has created more revisions and new files, and how developers are collaborating or working separated from one another

The tool has been tested successfully with several open source projects in a lab environment by some expert users. The answers provided to our initial users by the proposed visual representations has been satisfactory.

VII. Acknowledgements

This work was supported by Spanish Government project TIN2010-21695-C02-01, by the Castilla y León Regional Government through GR47, by the Ministerio de Ciencia e Innovación of Spain under project FI2010-16234 and also by the Ministry of Science and Technology (MICIT) of Costa Rica.

References

[1] G. C. Murphy and D. Notkin, "Reengineering with reflection models: A case study," *IEEE Computer*, vol. 30, no. 8, pp. 29–36, 1997.

[2] D. A. Keim, F. Mansmann, J. Schneidewind, J. Thomas, and H. Ziegler, "Visual analytics: Scope and challenges," pp. 76–90, 2008.

[3] K. Puolamäki, A. Bertone, R. Théron, O. Huisman, J. Johansson, S. Miksch, P. Papapetrou, and S. Rinzivillo, *Mastering the Information Age Solving Problems with Visual Analytics*. Eurographics Association, 2010, ch. Data Mining, pp. 39–56.

[4] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. J. van Wijk, and A. van Deursen, "Understanding execution traces using massive sequence and circular bundle views," in *Proceedings of the 15th IEEE International Conference on Program Comprehension*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 49–58.

[5] J. García, A. González Torres, D. A. Gómez Aguilar, R. Therón, and F. J. García Peñalvo, "A visual analytics tool for software project structure and relationships among classes," in *Proceedings of the 10th International Symposium on Smart Graphics*, ser. SG '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 203–212.

[6] M. Pinzger, H. Gall, M. Fischer, and M. Lanza, "Visualizing multiple evolution metrics," in *Proceedings of the 2005 ACM symposium on Software visualization*, ser. SoftVis '05. New York, NY, USA: ACM, 2005, pp. 67–75.

[7] L. Voinea and A. Telea, "Cvsgrab: Mining the history of large software projects," in *EUROVIS - Eurographics /IEEE VGTC Symposium on Visualization*, 2006.